Tutorial 1: paradoxes

We discuss the 'paradox' of the P_n's on pages 4-5 of the notes. Recall that P_0, P_1,... are all the programs of our Modula_2-like programming language, in alphabetical order. The program P was:

```
repeat forever
    generate the next program $P_n$ in the list
    run $P_n$ as far as the $n^{th}$ bit of the output
    if $P_n$ terminated or prompted for input before the $n^{th}$ bit was output then
        output 1
    else if the $n^{th}$ bit of $P_n$'s output is 0 then
        output 1
    else if the $n^{th}$ bit of $P_n$'s output is 1 then
        output 0
 end if
 end repeat
```

1. P must be some P_n, but for any n, its output differs from P_n at the n^{th} bit. This is impossible. What is wrong with our reasoning?
2. Suppose $P = P_{19}$ (say). What would happen on the 19th loop of P?
3. Suppose that H(x) is a procedure in our language, having the following property: for any program Q (supplied as a text string), H(Q) = 1 if Q halts when run, and H(Q) = 0 otherwise.
   i) Modify P using H to obtain a genuinely paradoxical program. [Hint: use H(run P_n as far as the n^{th} bit of the output).]
   ii) Deduce that H does not exist.
4. i) What is the least number that is not the answer to an English question having fewer than 200 letters?
   ii) C.C. Chang and H.J. Keisler kindly dedicated their book 'Model Theory' to all those people who haven't got a book dedicated to them. Is it dedicated to you or not?
   iii) What are the implications for your reasoning powers, if the following sentence is (a) true, or (b) false? "The reader has no way of convincing him/herself that this sentence is true."

Tutorial 2: basic Turing machines

1. Design a TM Z whose output is "hello world" on any input. So formally, the input/output function f_Z of Z is: $f_Z(w)$ = hello world for any word w of Z's input alphabet (say, the roman alphabet).
2. (a) Design a Turing machine M to solve the following problem. The input is a word of {1} of length n. So before M starts its run, its tape contains a 1 in squares 0, 1, …, n-1, and blanks ($\wedge$) in all other squares. The problem is to determine whether n is even or odd. When M halts, square 0 of the tape should contain a 0 if n is even, and a 1 if n is odd. Square 1 should contain a blank.
   (b) Write down the input/output function of M in mathematical notation.
3. Describe a TM, R, that reverses a word of {0,1}. So $f_R(00011)$ = 11000, etc. Use ‘pseudocode’ or English description if possible.

Tutorial 3: arithmetic, multi-tape Turing machines

If $n \geq 0$ is a number let 'n' $\in \{0,1\}^*$ be the binary expansion of n, without leading zeros, written for your convenience with the least significant digits on the left.

So eg. '8' = 0001 $\in \{0,1\}^*$, and '0' = 0.

1. Design a 2-tape Turing machine A with f_A('n','m') = 'n+m' for all n, $m \geq 0$ (so A adds two numbers in binary). You can cheat and assume that initially 'n' is on tape 1, and 'm' on tape 2. State the input and full alphabets of A.
2. Explain briefly how to design a (multi-tape) Turing machine M such that f_M('n','m') = 'n.m'. That is, M multiplies two binary numbers. [E.g., remember that multiplication is repeated addition; you can use A as a subroutine.]
3. A *palindrome* (of $\{0,1\}^*$) is a word $w = s_1 s_2 \ldots s_n$ in $\{0,1\}^*$ such that $w = s_n s_{n-1} \ldots s_1$. Eg. 0110, 010, 0 and ε are palindromes; 011 is not.
   Design a flowchart or give pseudocode for a Turing machine M with input alphabet I = {0,1} such that for all $w \in I^*$, M halts and succeeds on w iff w is a palindrome. [Hint: use 2 tapes.] What is the full alphabet of your M?

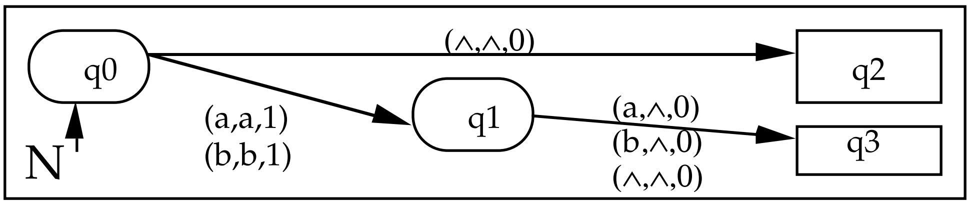
Tutorial 4: more multi-tape Turing machines, & UTM

1. Design a Turing machine M with input alphabet {a,b}, which, given as input a word w of this alphabet, outputs the word obtained from w by writing out its a's, and then its b's, in order. For example,

$$f_M(babaa) = aaabb$$

You may use pseudo-code or a flow-chart diagram; in the latter case you should explain your notation for instructions. You may use several tapes, and you can assume that square 0 of each tape is implicitly marked.

2. (a) What is meant by the term "<u>standard</u> Turing machine"?
(b) What does the following standard Turing machine N do (i.e., what is its input/output function)?



(c) Write out its code.

3. The universal Turing machine U can be made a standard Turing machine. So can M and N above [asssume M halts & fails if its input word is not in {a,b}). So they have codes, namely code(U), code(M), code(N).
Calculate:
- f_U(code(U)*code(M)*babba)
- f_U(code(U)*code(N)*) [not a misprint!]
- f_U(code(U)*code(U)*code(N)*c)

Tutorial 5: Halting problem

The halting problem is very important. If you are not sure about it, please ask a tutor. Then try these questions. Remember the three Ds.[1]

1. Show that there is no Turing machine H' such that for all standard Turing machines S and words w of C:

$$f_{H'}(\text{code}(S)*w) = \begin{cases} 1 \text{ if S halts on input w;} \\ 0 \text{ otherwise} \end{cases}$$

   (The 'usual' machine H prints 1 if S <u>halts and succeeds</u> on w.)

2. Show that there is no Turing machine EO (standing for "empty output") such that for all standard Turing machines S and words w of C:

$$f_{EO}(\text{code}(S)*w) = \begin{cases} y \text{ if } f_S(w)=\varepsilon; \\ n \text{ otherwise} \end{cases}$$

3. Show that there is no Turing machine H" such that for all standard Turing machines S and words w of C:

$$f_{H''}(\text{code}(S)) = \begin{cases} 1 \ \text{ if S halts \& succeeds on input code(S);} \\ 0 \ \text{ otherwise} \end{cases}$$

   (The 'usual' machine H takes input code(S)*w, i.e., 2 items.)

4. Can you show there is no H" as in (3) by using the method of reduction: reducing HP to the problem that H" solves?

[1] Be Defiant, and Don't forget about the Duplicator.

2.8=mc2.8 Tutorial 6: reduction

Questions 1,2,5 use the machine M[w], EDIT, and the universal Turing machine U; all are assumed to be standard. See §4.3 and §5.3 of the notes.

1. Let R be a standard Turing machine that reverses its input. So, for example, $f_R(abc) = cba$. Evaluate:

a) $f_R(alucard)$; b) $f_{R[Lee]}(Cushing)$; c) $f_{R[marb]}(\varepsilon)$

2. Let S be any standard Turing machine, and let v, w be any words of C. Check that $f_U(f_{EDIT}(code(S)*v)*w) = f_S(v)$.

3. **Uniform halting problem** (the problem of whether a Turing machine halts on every input). Show by reduction that there's no Turing machine UH such that for all standard Turing machines S,

$$f_{UH}(code(S)) = \begin{cases} y & \text{if S halts \& succeeds \underline{on every input}} \\ n & \text{otherwise} \end{cases}$$

[Adapt the argument for the empty input halting problem.]

4. **Inverse problem** Show by reduction that there's no Turing machine INV such that for all standard Turing machines S_1, S_2, and all words w of C,

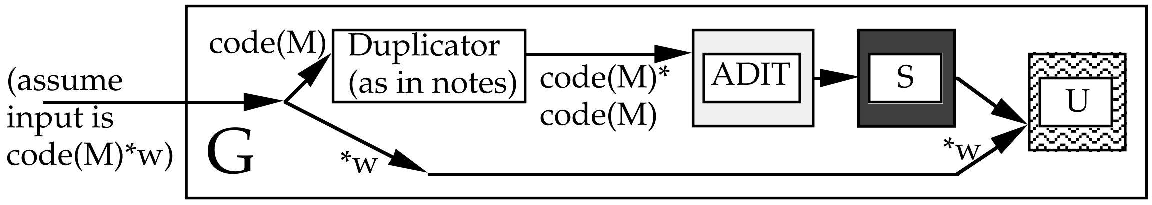
$$f_{INV}(code(S_1)*code(S_2)*w) = \begin{cases} da & \text{if } f_{S_2}(f_{S_1}(w))=w \\ nyet & \text{otherwise} \end{cases}$$

That is, on input w, S_2 inverts the effect of S_1. Note that $f_{S_2}(f_{S_1}(w))$ is undefined if $f_{S_1}(w)$ is, so $f_{INV}(code(S_1)*code(S_2)*w)$ = nyet in this case.

[You might try to transform an instance code(S)*w of HP into code(S)*code(W)*w, for some suitable Turing machine W. You may have to think a bit about what W is to do.]

5. ['Recursion theorem', Stephen Kleene, 1938] The Turing machine ADIT is similar to EDIT (§5.3), but, on input code(M)*w, it outputs code(M{w}). M{w} is a standard TM similar to M[w], but, on input x, it writes w* in front of x (shifting x right to make room!), then runs M. So $f_{M\{w\}}(x) = f_M(w*x)$.

Let S be any standard TM, and consider the TM G below.



G is assumed standard by scratch character elimination. On input code(M)*w, G splits off code(M), duplicates it, and runs ADIT, then S; G then tacks on *w to the output, and runs U. So $f_G(code(M)*w) = f_U(f_S(f_{ADIT}(f_{Duplicator}(code(M))))*w)$.

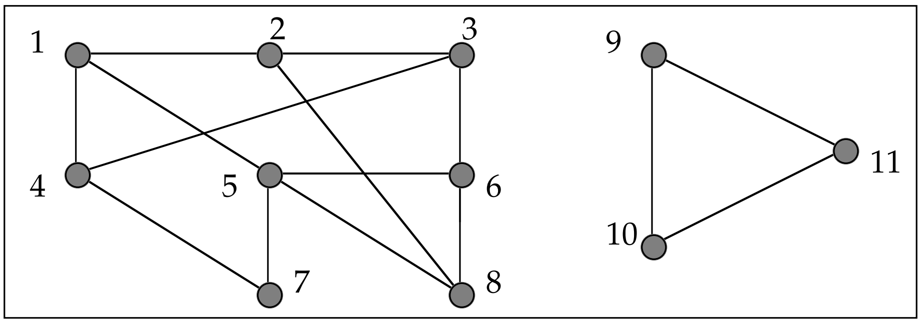
(a) Let K = G{code(G)}. Check that for any word w of C, we have

$$f_K(w) = f_U(f_S(code(K))*w).$$

(b) Deduce that, given <u>any</u> algorithm to modify TMs (i.e., their codes) in any way whatever, there's always some TM whose input-output function is <u>unchanged</u> by the modification.

2.8=mc2.8 Tutorial 7: graphs

1. Consider the following graph:



(a) How many connected components are there?

(b) Now use the algorithm of §7 (below) on this graph in (i) depth-first mode (priority = current time), and (ii) breadth-first mode (priority = 12 – current time). What are the fringe contents at each stage of execution in the depth first case?

```
1  Visited(n): global Boolean array, initially all false; x: integer
2  repeat with x=1 to n
3      if not visited(x) then visit(x)
4  end repeat
5  procedure visit(x)
6      x,y,z: integer                --- to represent vertices
7      empty the fringe (priority queue)
8      push x into fringe, with label *, and any priority
9      repeat until fringe is empty
10         pop (x,y,p) from fringe  [So x was the queue entry; y was
               its label; and p was the (highest possible) priority.]
11         visited(x) := true
12         repeat for all nodes z connected to x by an edge
13             if not visited(z) then
14                 push z into fringe, with label x, and chosen priority
15             end if
16         end repeat
17     end repeat
18 end visit
```

(c) Which spanning trees for the connected components do you come up with in each case?

(d) How many times is *visit* called (line 3) in each case? Is it the same no. as (a)?

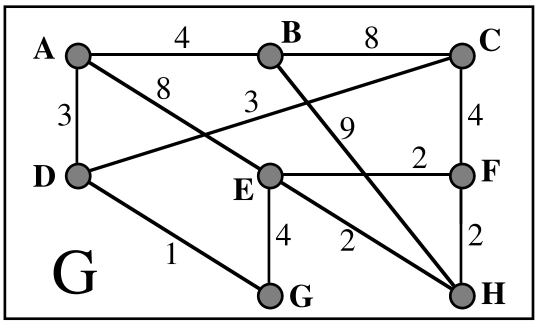
(e) Give a value of x for which visited(z) is true more than once during the loop in lines 12–16. What is the significance of this?

2. Show that any graph with n nodes and < n-1 edges (for any n≥1) is not connected.

3. Show that any <u>connected</u> graph with n nodes and n-1 edges (any n≥1) is a tree. Find a graph with n nodes and n-1 edges (some n) that's <u>not</u> a tree.

Tutorial 8: weighted graphs

1. Run Prim's algorithm to find a minimal spanning tree of the following weighted graph G (starting at node A). What tree do you get? What is its total length?



2. Run it <u>again</u>, starting somewhere else. Do you get the same total length of tree? Or the same tree?!

3. Let's check that <u>our algorithm finds a MST even in a graph where some edges have equal weight</u>. We can use G = (V,E,w) above, as some of its edges have equal weight. <u>The idea is to disturb the weights in G just a little, enough to make all the edges different, so that the algorithm definitely works, but not enough to alter the MSTs.</u> Follow these easy steps:

(i) <u>Lengthen</u> each of the 12 edges of G a little, so that:

  (a) for each n = 1,…,7, the nth edge chosen by the algorithm (when you ran it in Q1) is lengthened by n/100 (i.e., by 0.01, 0.02, …, 0.07);

  (b) the numbers 0.08, 0.09, 0.10, 0.11 and 0.12 are added randomly to the remaining 5 edges (a different number to each!).

You get a new weighted graph G* = (V,E,w*), with the same nodes and edges as G, but slightly different weights. No edge is altered by more than 0.12: we have $w^*(e) \le w(e)+0.12$ for all $e \in E$.

(ii) Let (V,T) be <u>any</u> spanning tree of G. (The nodes are V, the edges $T \subseteq E$.)

  (a) How many edges does T have?

  (b) Let $m = \sum_{t\in T} w(t)$, and $m^* = \sum_{t\in T} w^*(t)$, be the total weights of the edges of T, measured in G, G*, respectively. Show that $m \le m^* < m+1$.

(iii) Use Prim's algorithm (starting at node A) to find a MST of G*. Call it T. As all edges of G* are different in length, T really is a MST of G*.

(iv) <u>Why</u> did you get the same tree T as in Q1? (I hope you did!)

(v) We know that T is a MST of G*. <u>Show that T is also a MST of G</u>. [Use (ii).] Deduce that the algorithm finds a MST even in graphs with some equal edges (or at least on G!!)

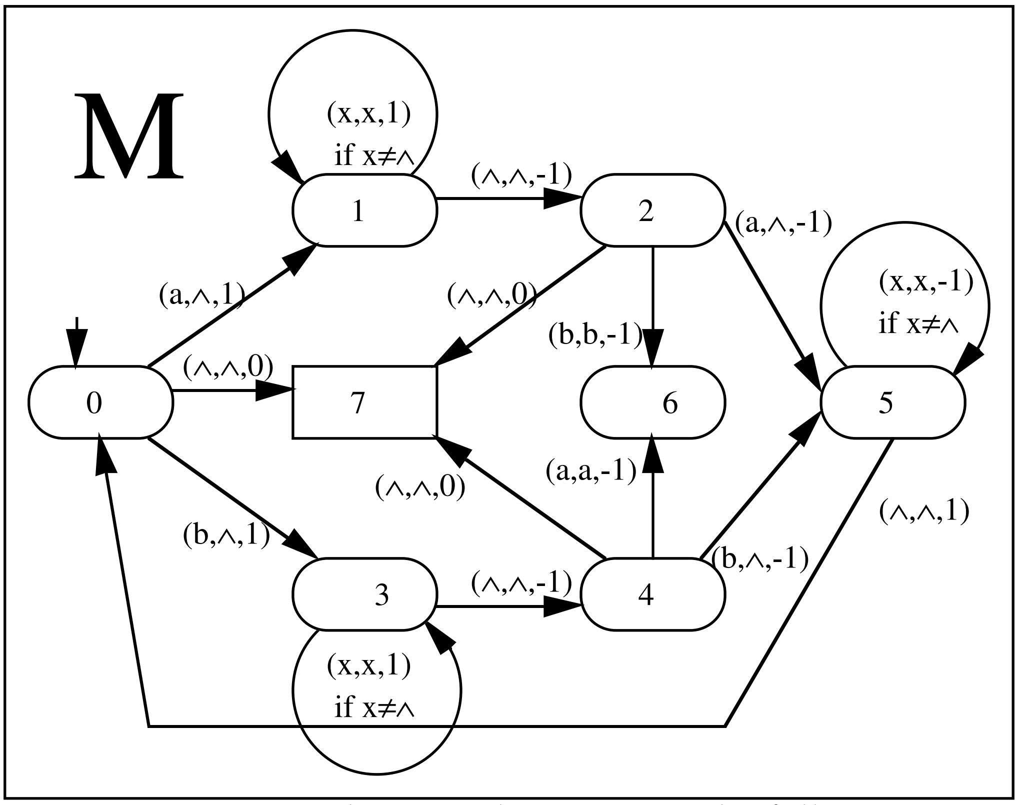
(vi) If G had 100 or 1,000 nodes, say, ii(b) would not be true any more. Why? How would we alter our proof in such a case?

(vii) In i(a) we divided n by 100. Find the smallest divisor that would have done.

PROBLEM: use the separation property to show that the algorithm works on any weighted graph.

Tutorial 9: basic complexity theory

1.(i) What does the Turing machine M do? (Its input alphabet is {a,b}.)



(ii) How many steps does M take to run on the following inputs:
- abab
- abaa
- abba

(iii) What are the worst case inputs? Give one of length 4. And of length 5.

(iv) Show that

$$\text{time}_M(n) = (n^2 + 3n + 2)/2 \quad \text{for all n,}$$

so that M runs in polynomial (quadratic) time. [E.g., find an equation for $\text{time}_M(n)$ in terms of $\text{time}_M(n-2)$, and use induction on n. The base cases of even- and odd-length inputs are different.]

2. Design (flowchart or pseudo-code) a 2-tape Turing machine M_2 that is equivalent to M above, and runs in <u>linear</u> time (i.e., for some constants a, b, we have $\text{time}_{M_2}(n) \leq a + bn$ for all n). Check that it does run in linear time.

3. There is a deterministic Turing machine BU that, given the binary representation of a number as input, outputs the unary representation. Show that BU <u>can't</u> have polynomial time complexity.
[Hint: how long does BU take to output the answer if the input is the binary representation of n?]

Extra tutorial: miscellaneous

A single exam style question (35 min) might consist of 1 and 2 together.

1. Using *either* pseudocode *or* a 'flowchart diagram', design a Turing machine M whose input alphabet I consists of the following three symbols:

] a [

such that M halts and succeeds on an input word w of I (M *accepts* w) if and only if w contains the same number of left brackets ' [' as right brackets '] '.

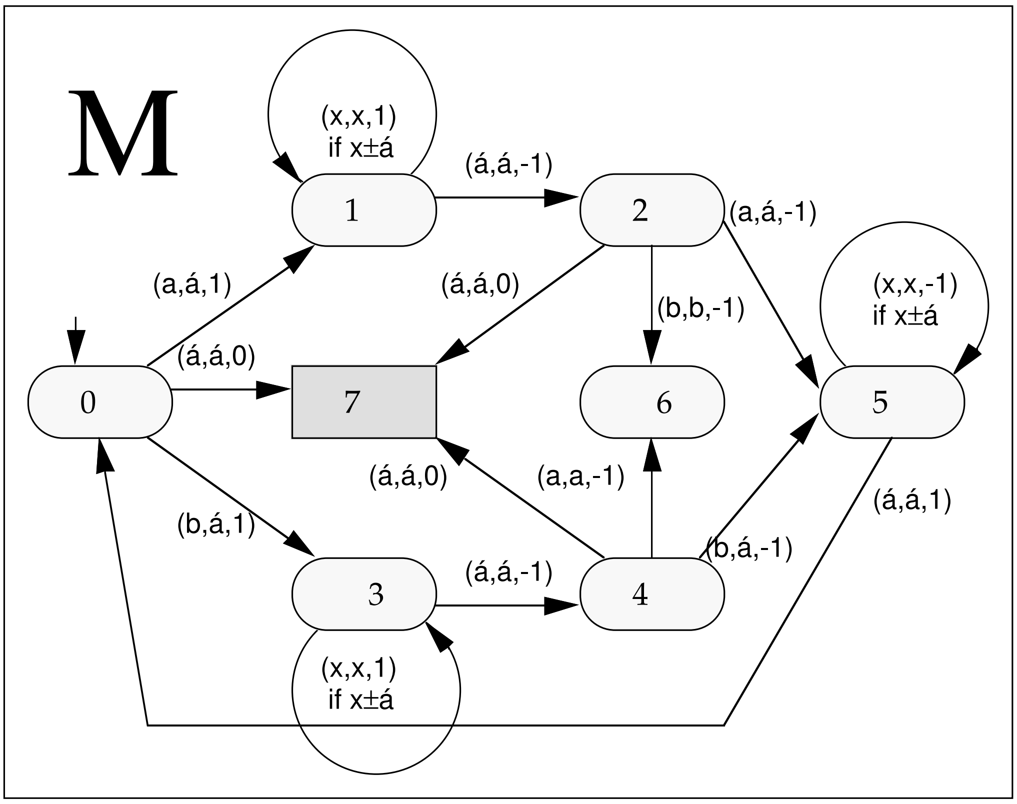
Example: the words [a a] and [ ] a]] a [[are accepted; the words [[and a [are rejected.

Hint: it may help to use three tapes. You may assume that square 0 of each tape is implicitly marked.

2. *Briefly* explain how you could modify M in (1) to make it accept exactly those words w = $a_1a_2...a_n$ such that:

- w has the same number of left and right brackets, and
- for all $1≤i≤n$, $a_1a_2...a_i$ has at least as many left brackets as right brackets. [E.g., a well-formed formula of logic has brackets like this.]

3. What does the following standard Turing machine M do? (It has input alphabet {a,b} and full alphabet {a,b,∧}.) What is its input-output function? How could we simplify the diagram using a parameter x ∈ {a,b}? Write out *code*(M) (see §4.2.1 of notes).


M
0
1
2
3
4
5
6
7
(x,x,1) if x±á
(á,á,-1)
(a,á,-1)
(a,á,1)
(á,á,0)
(b,b,-1)
(x,x,-1) if x±á
(á,á,0)
(á,á,0)
(a,a,-1)
(á,á,1)
(b,á,1)
(á,á,-1)
(b,á,-1)
(x,x,1) if x±á